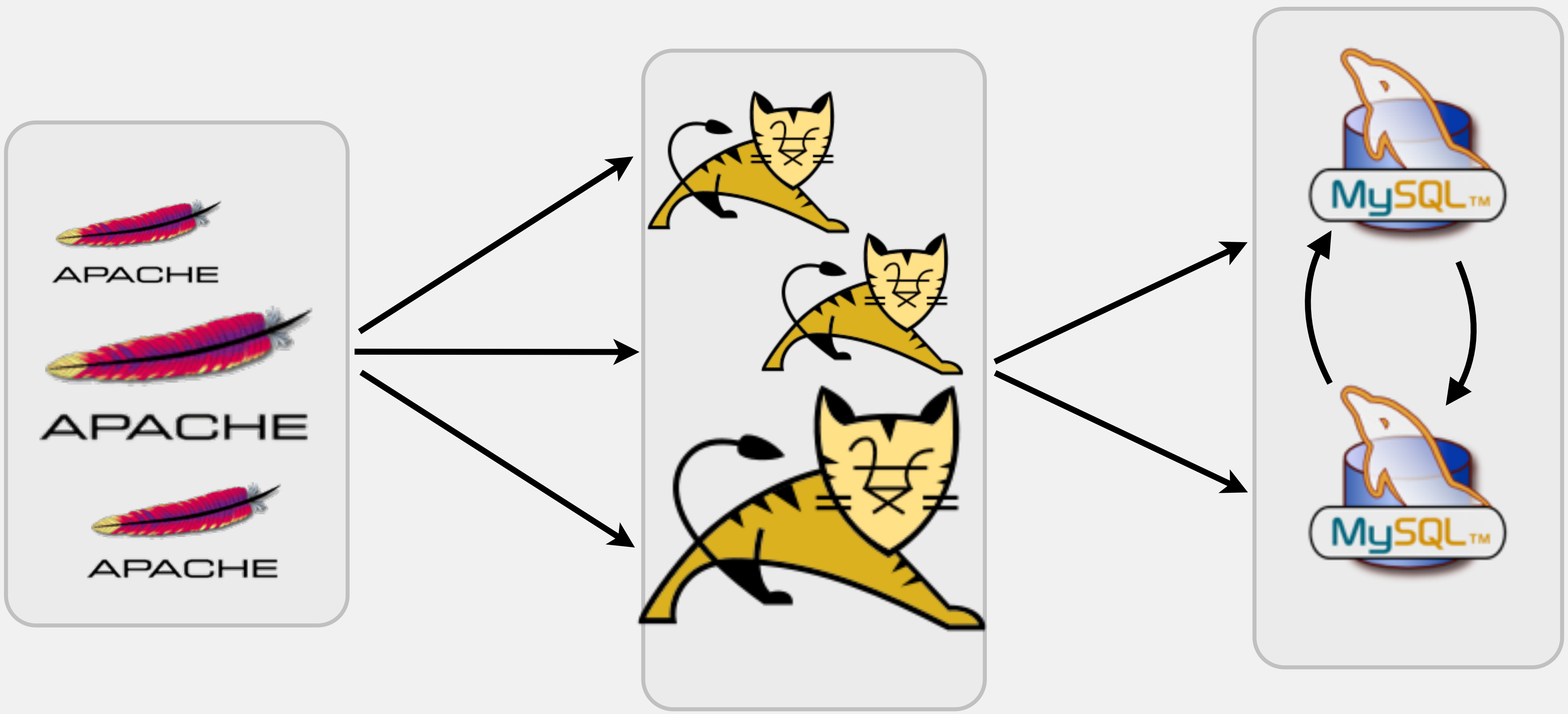
Higher SLA Satisfaction in Datacenters with Continuous Placement Constraints

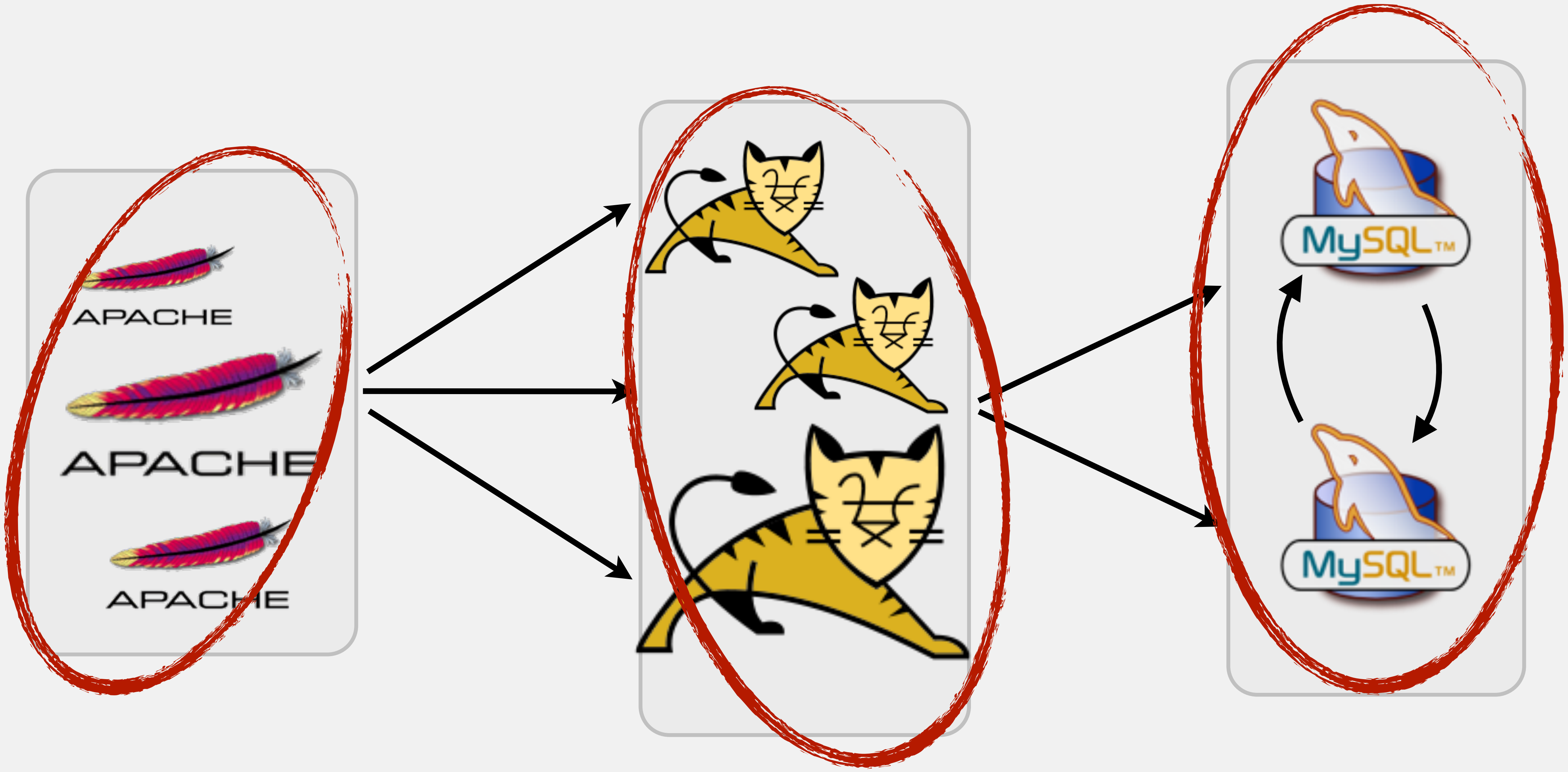
Huynh Tu Dang
hdang@polytech.unice.fr

Fabien Hermenier
fabien.hermenier@unice.fr

SLA for a virtualised application

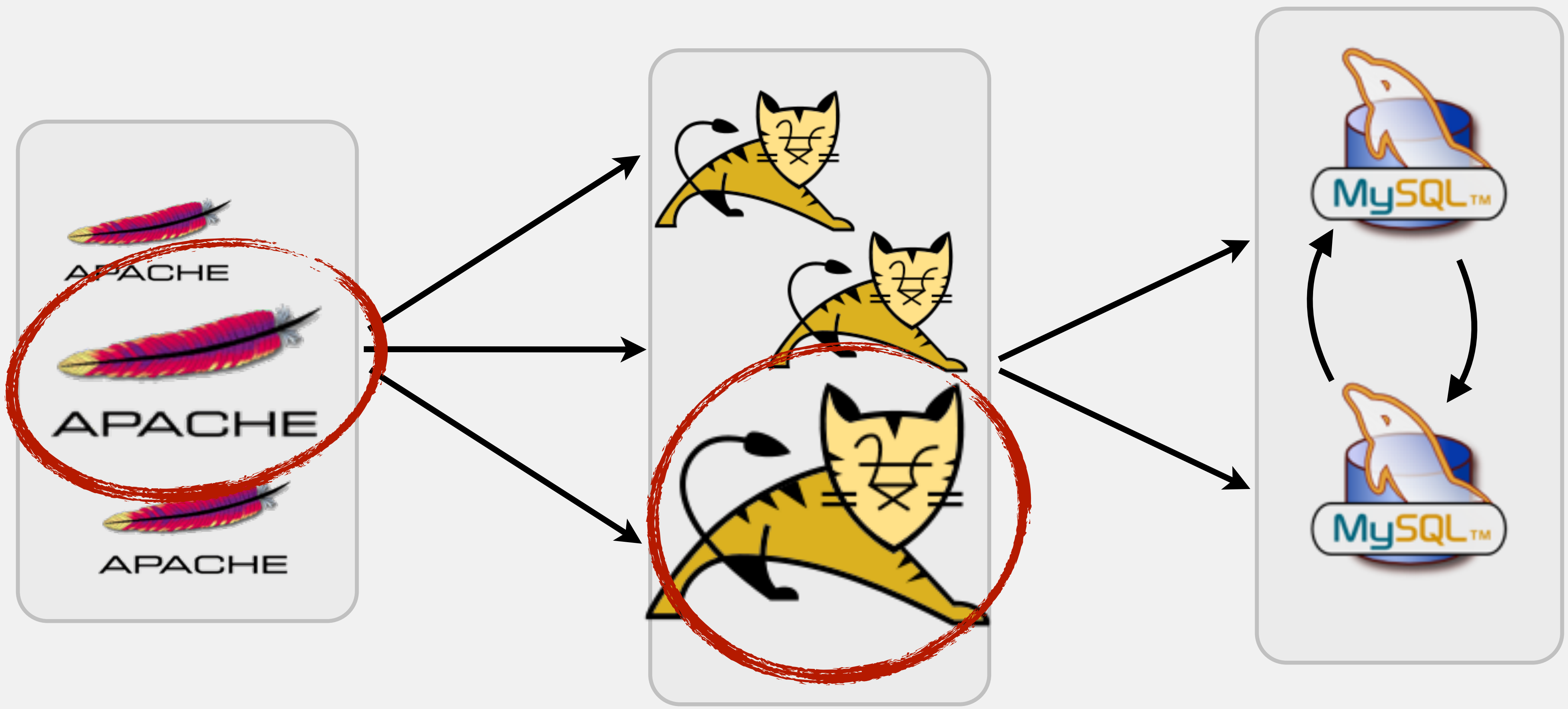


SLA for a virtualised application



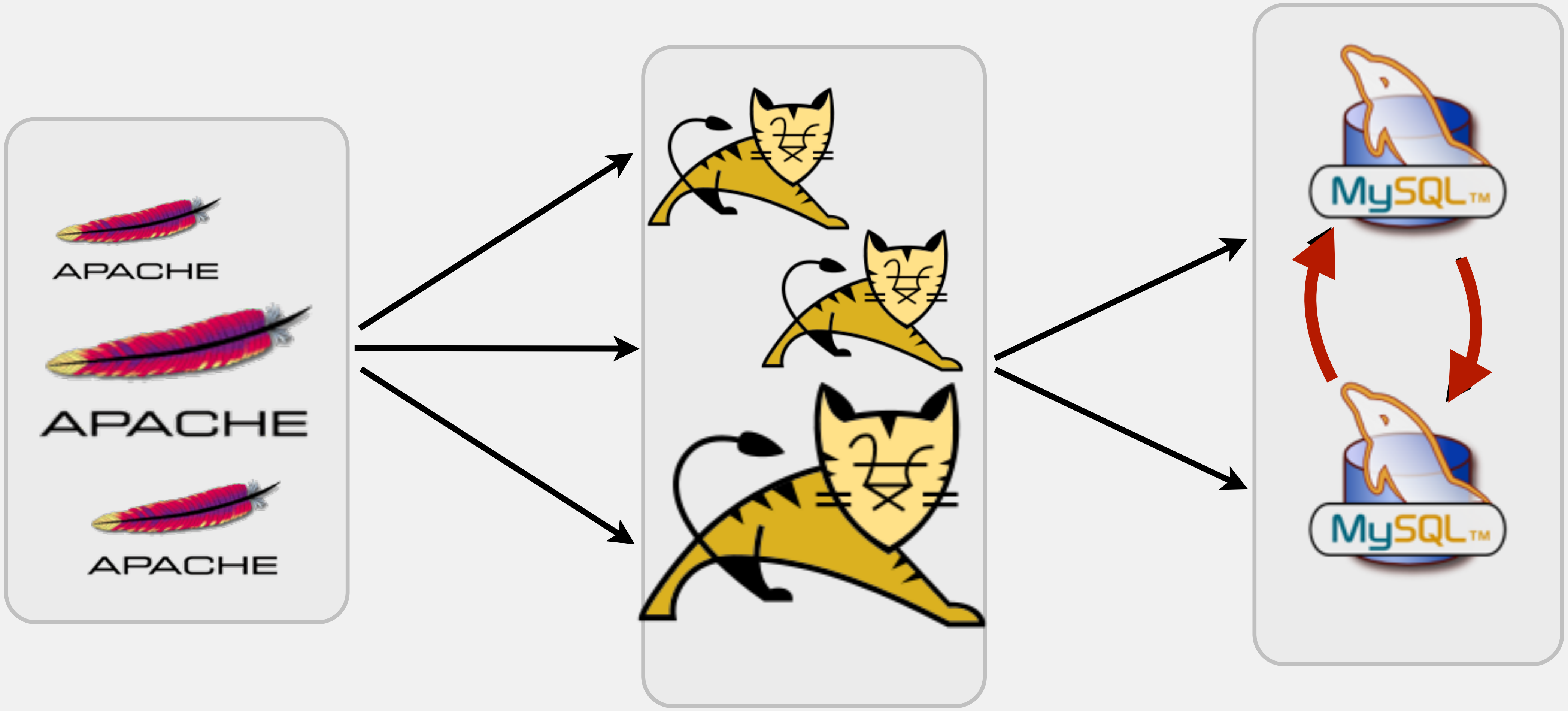
spread the replicas

SLA for a virtualised application



performance guarantee

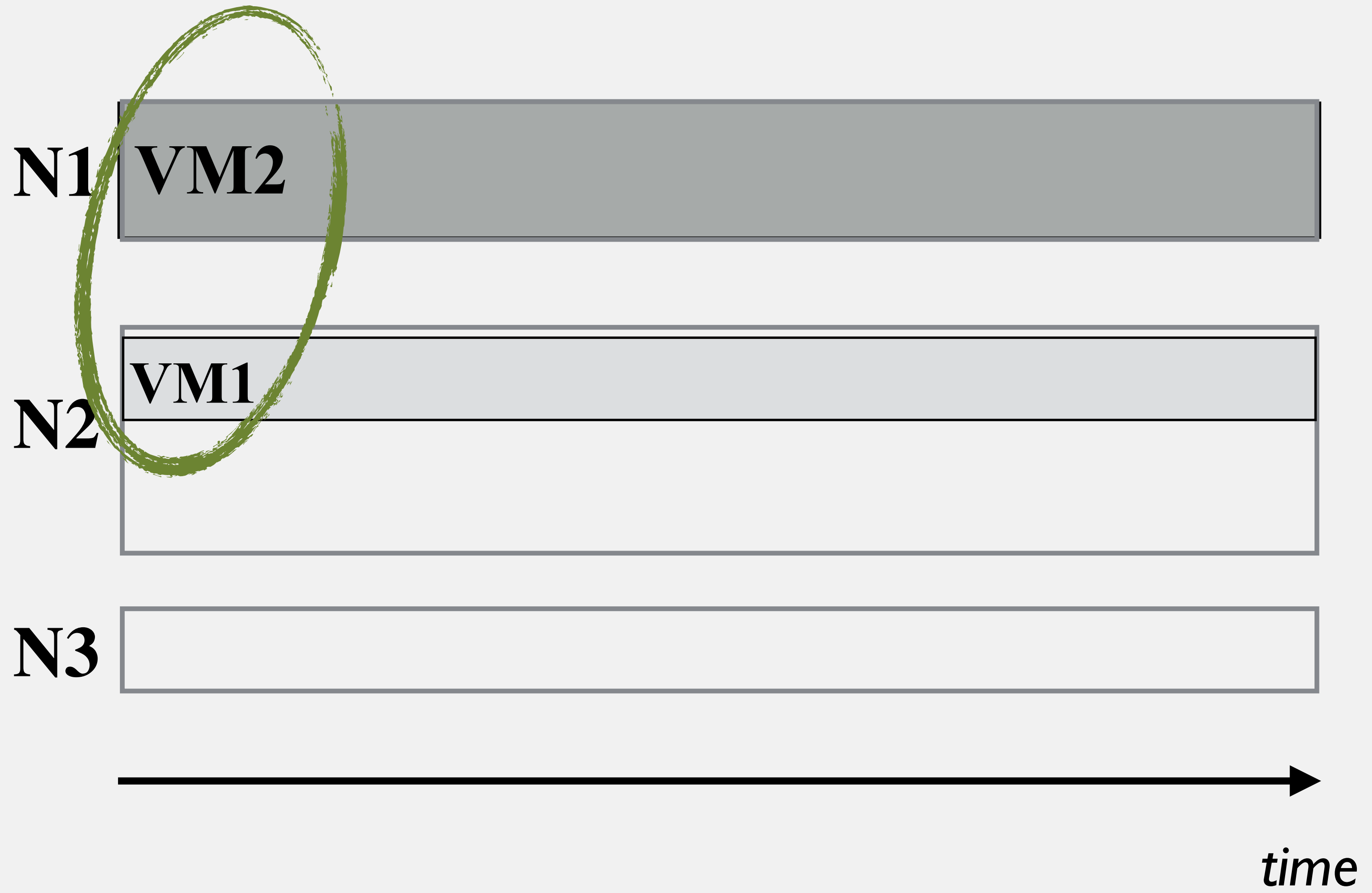
SLA for a virtualised application



low latency

reconfiguration algorithm

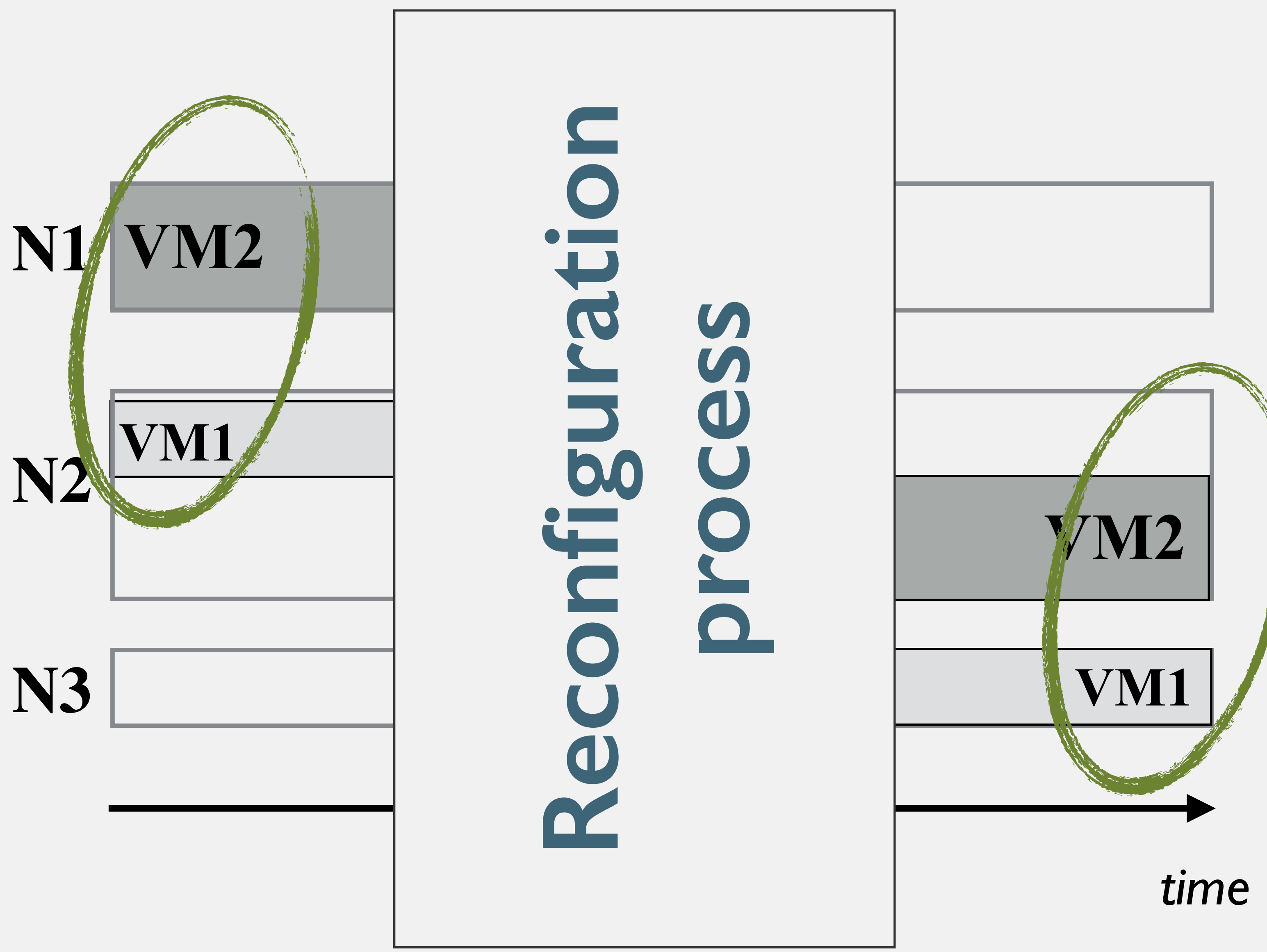
SLA: `spread(VM1, VM2)`



reconfiguration algorithm

SLA: `spread(VM1, VM2)`

sys-admin query: `offline(N1)`

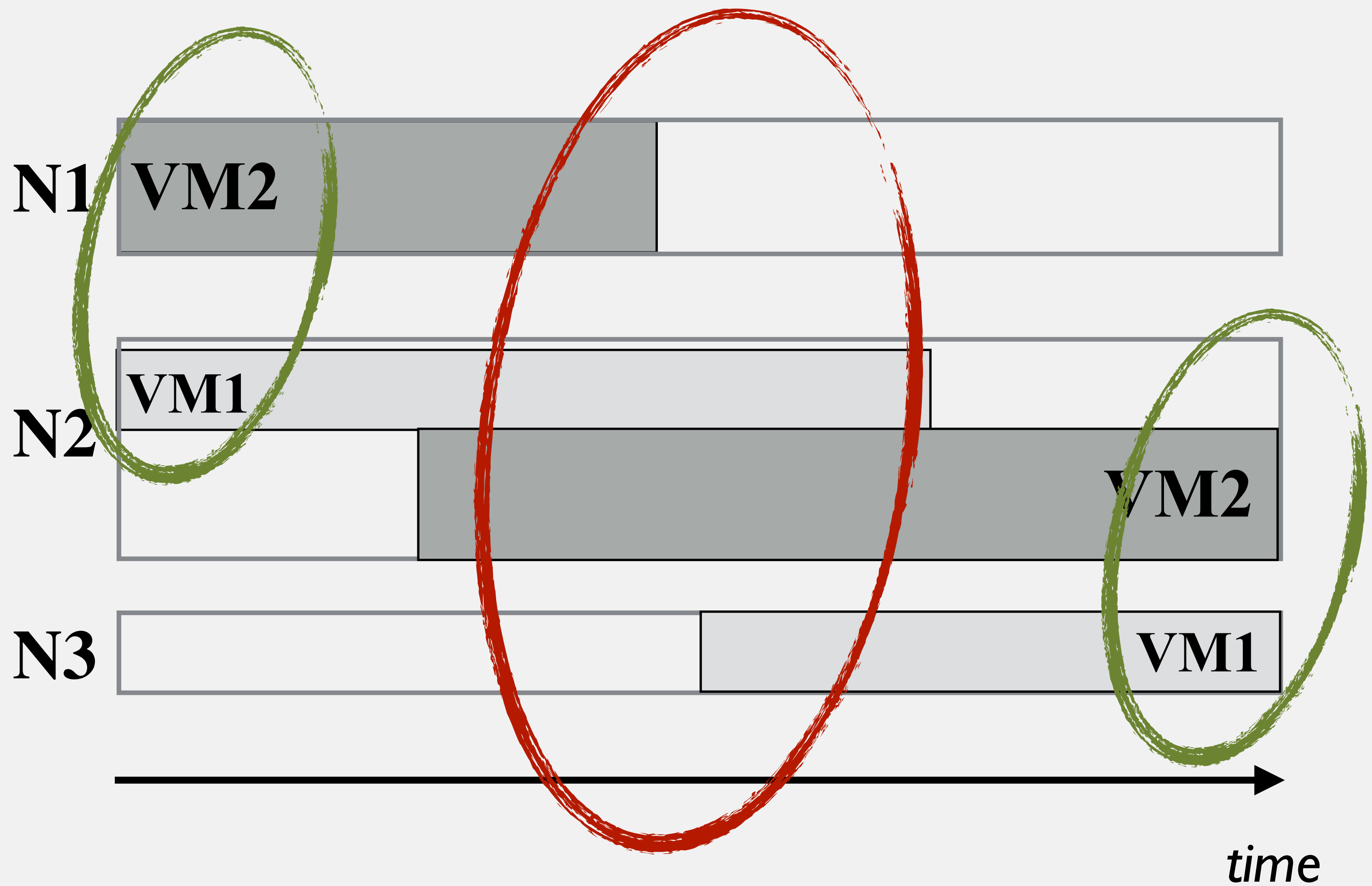


reconfiguration algorithm

with discrete restrictions

SLA: `spread(VM1, VM2)`

sys-admin query: `offline(N1)`



Discrete restriction is **not** enough

not an unpredictable situation,
an algorithmic issue

Evaluating the reliability of discrete placement constraints

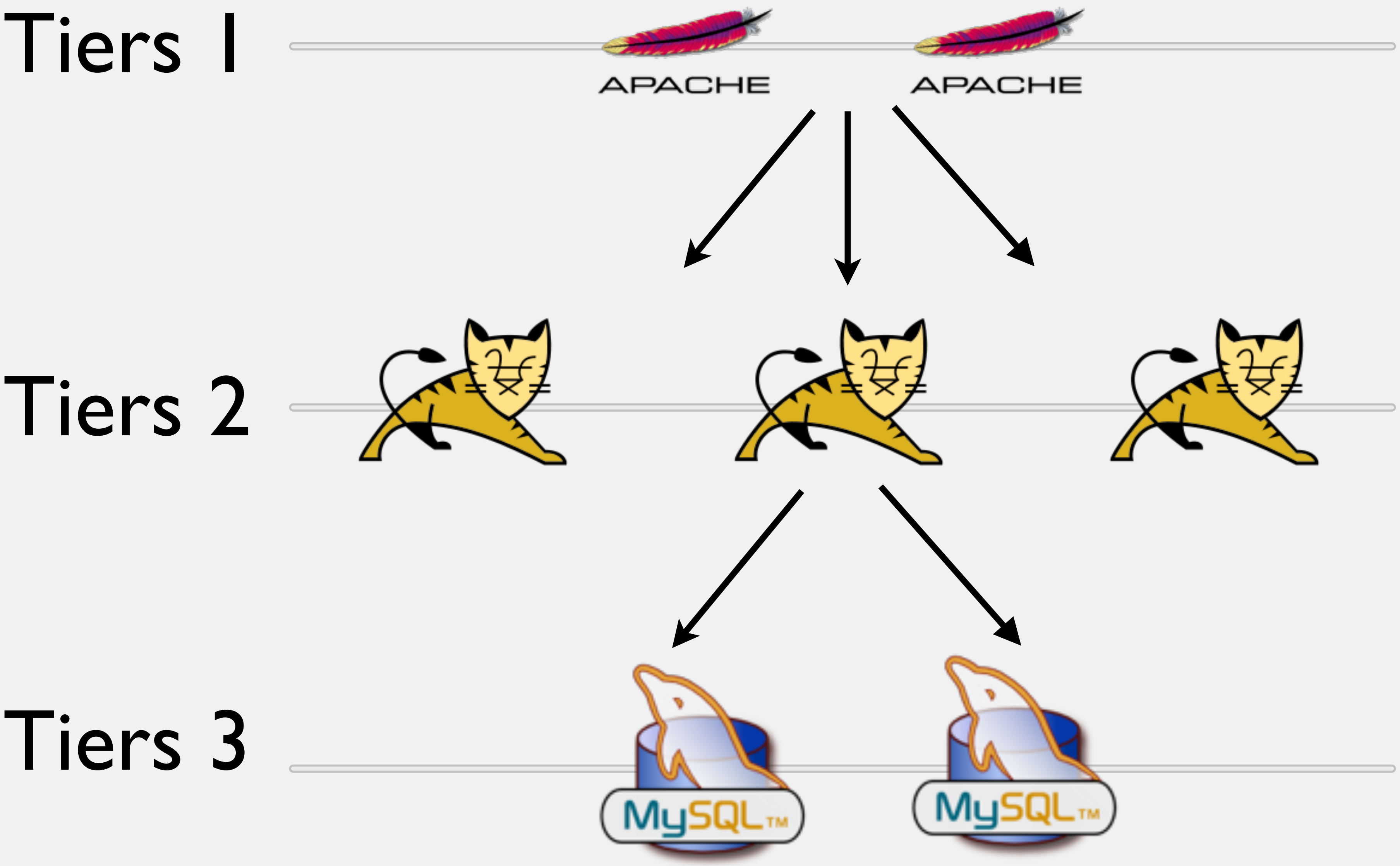
- **simulate** a 256-server datacenter
- running 350 **HA** webapp (5,200 VMs)
- **BtrPlace** as the reconfiguration algorithm
- 4 reconfiguration scenarios that mimic **industrial use case**
- **100** instances per scenario

Studied constraints

`spread`	replicas on distinct servers for fault tolerance
`among`	DBs on a same edge-switch for a fast synchronisation.
`splitAmong`	webapp split over 2 clusters for disaster recovery
`maxOnline`	240 nodes online at maximum to fit licensing policy
`singleResourceCapacity`	keep resource for hypervisor management operations

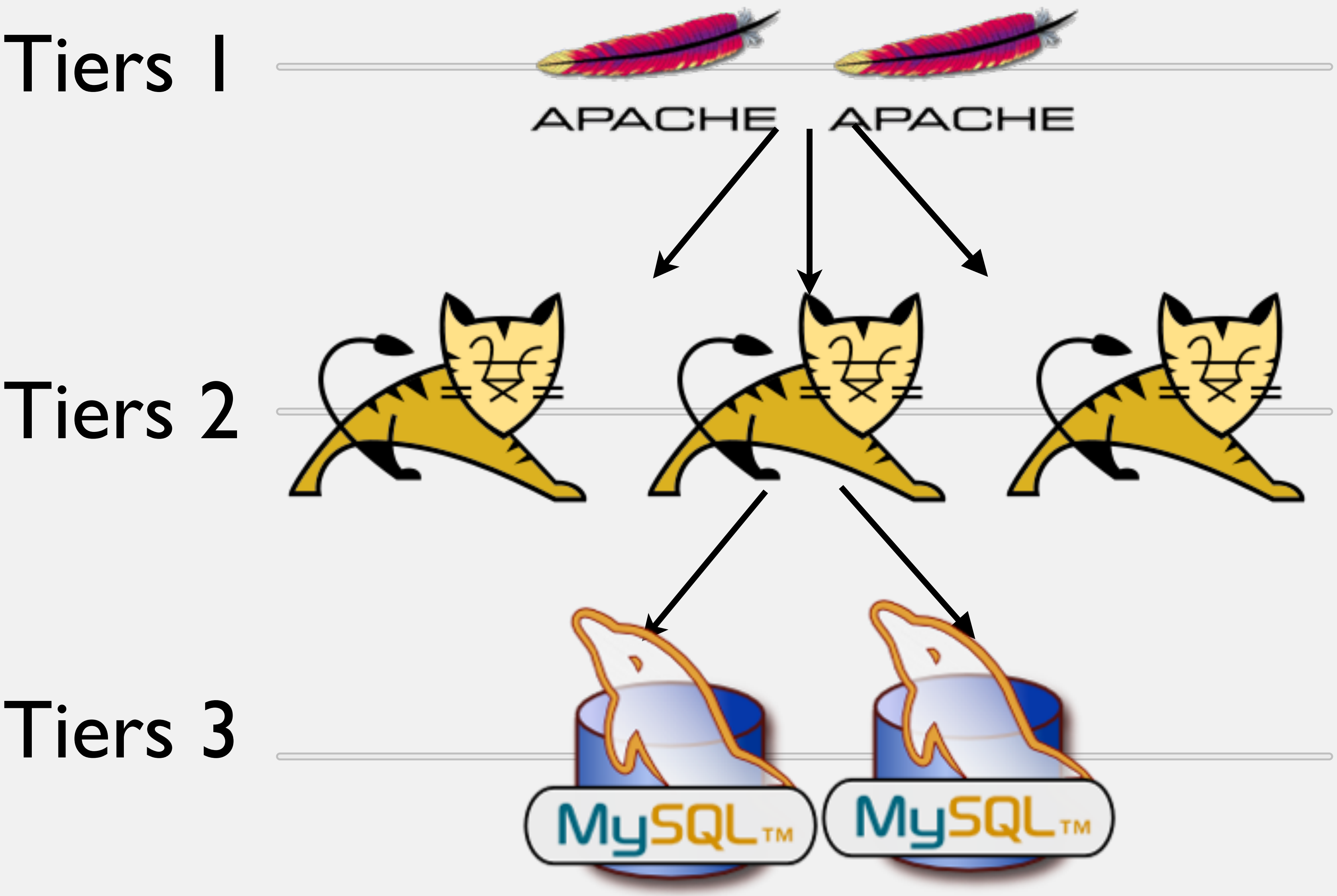
scenario

vertical elasticity



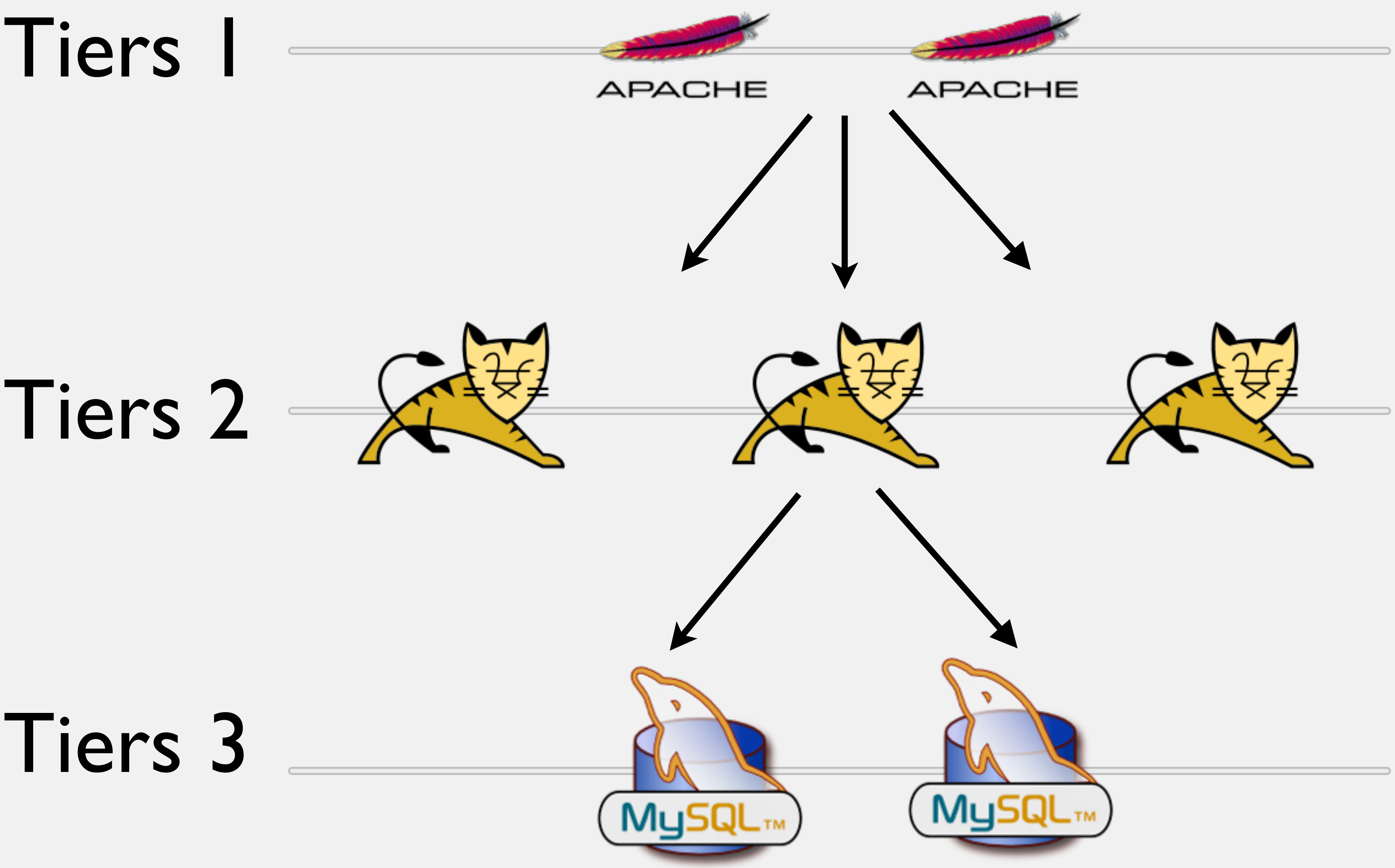
scenario

vertical elasticity



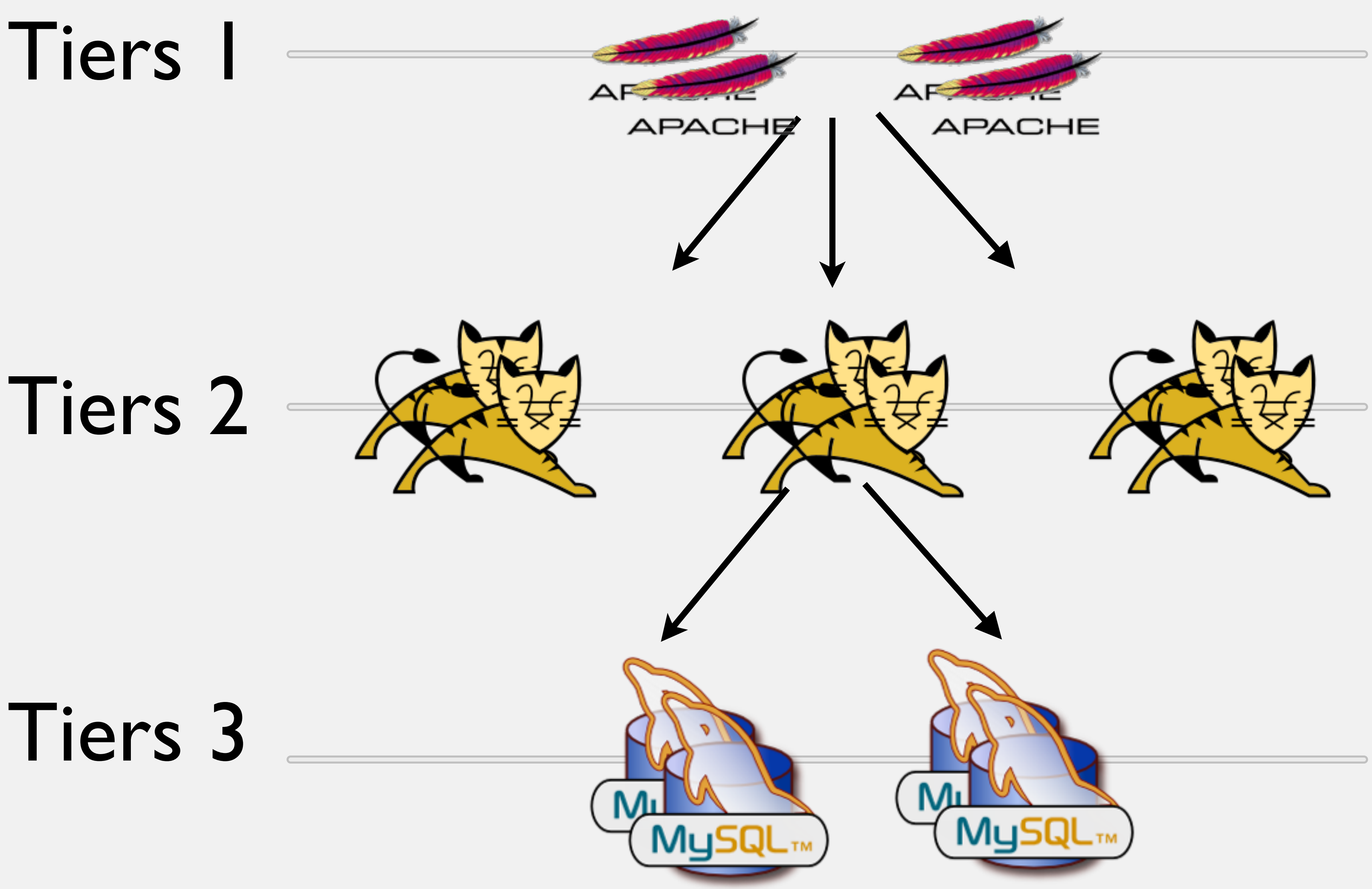
scenario

horizontal elasticity



scenario

horizontal elasticity



scenario

boot storm

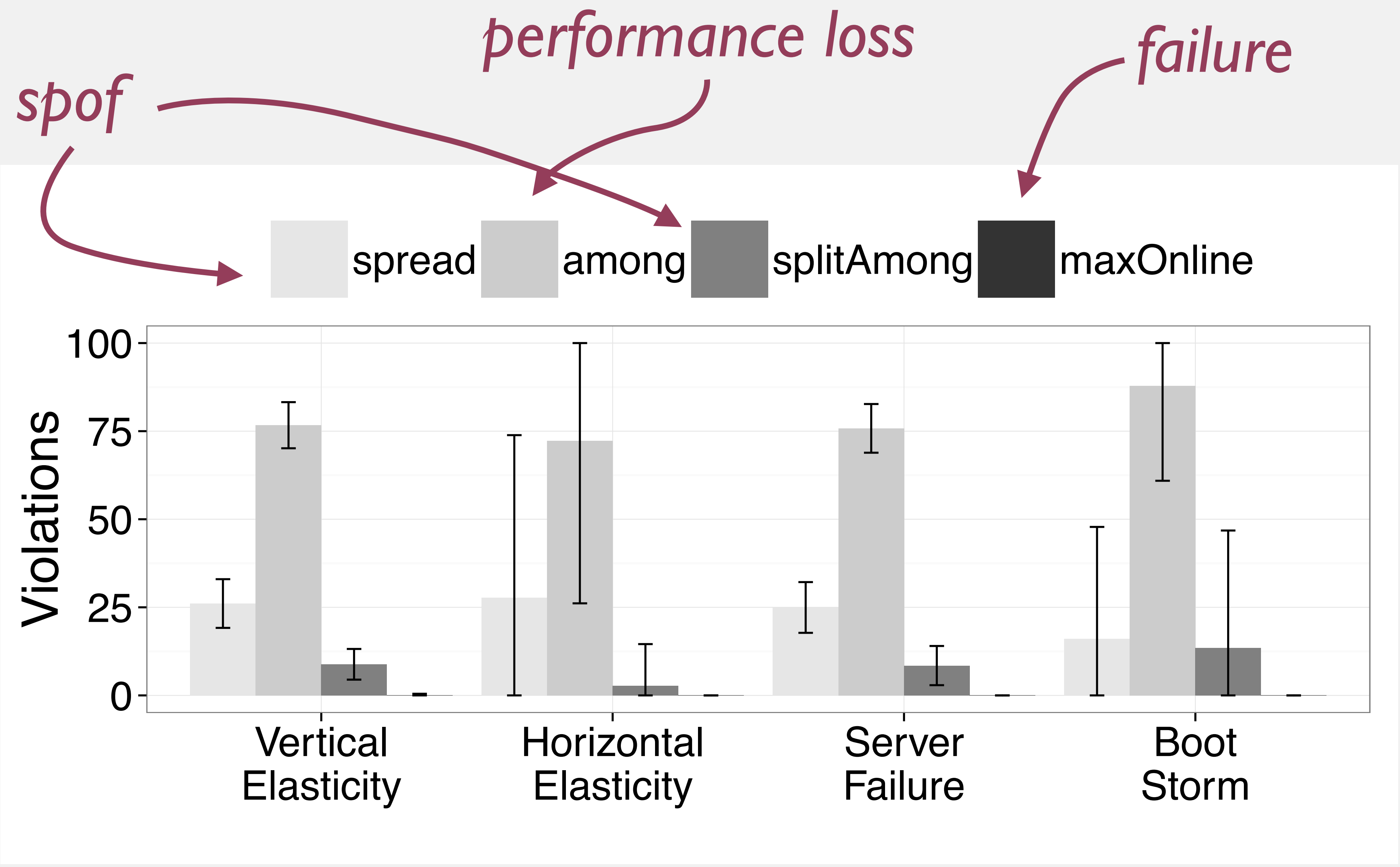
x 400

scenario
server failure

Migrations lead to unanticipated placements

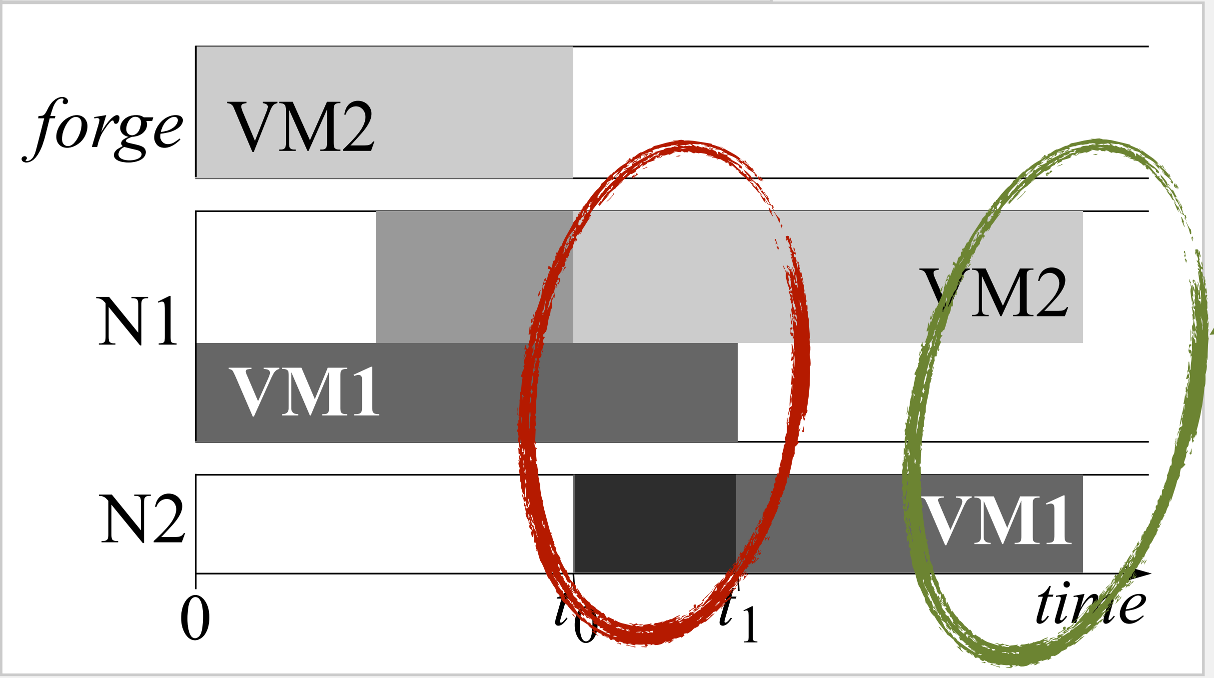
Scenario	Violated SLAs	Actions			
		VM Boot	Migrate	Node Boot	Node Shutdown
Vertical Elasticity	40.72	0%	99.99%	0.005%	0.005%
Horizontal Elasticity	0.19	99.82%	0.18%	2.82%	0%
Server Failure	29.56	61.29%	35.89%	2.82%	0%
Boot Storm	0.35	98.57%	1.43%	0%	0%

Migrations tend to violate relative placement constraints



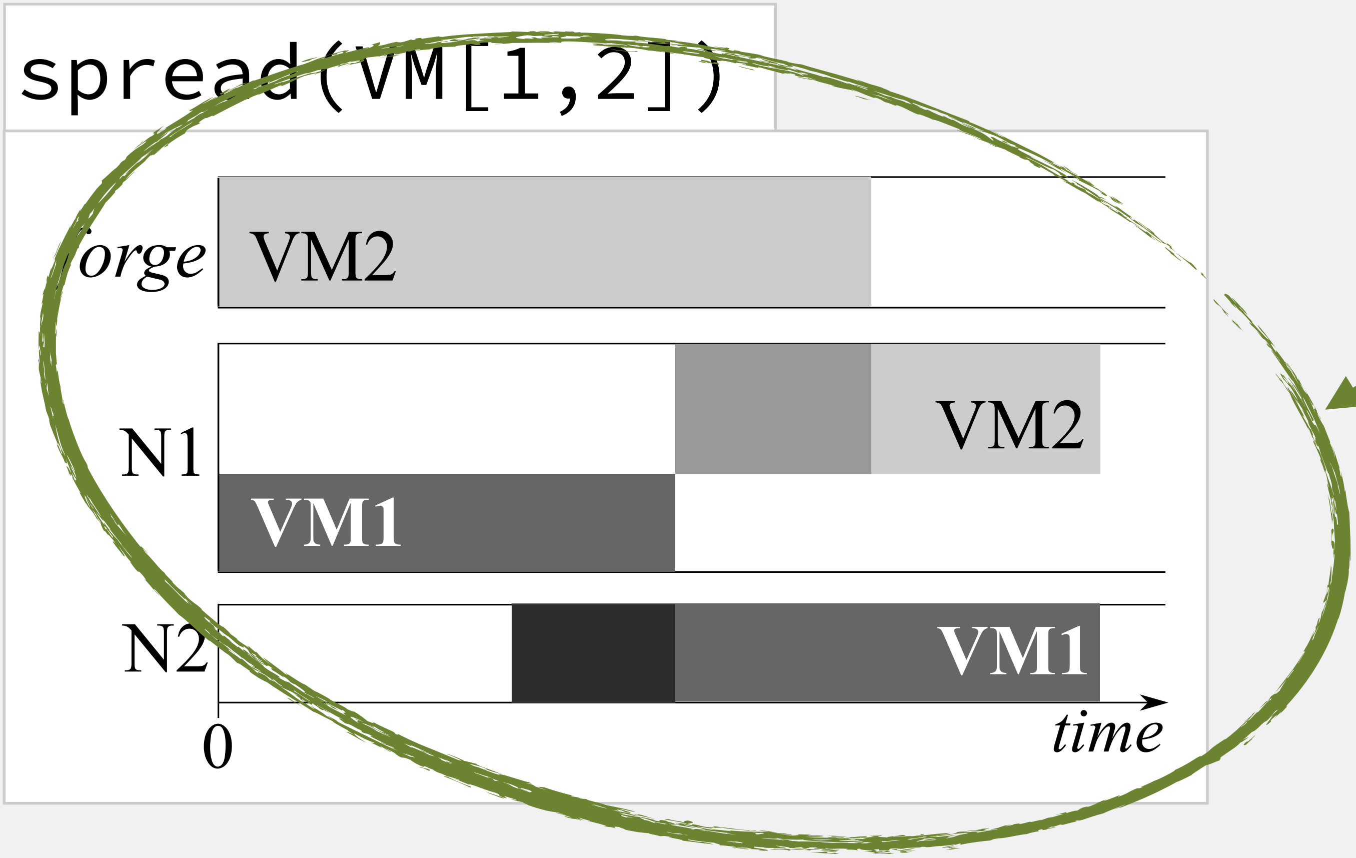
Trading **unreliable** discrete constraints ...

```
spread(VM[1,2])
```



we addressed an assignment problem

... for safe **continuous** constraints

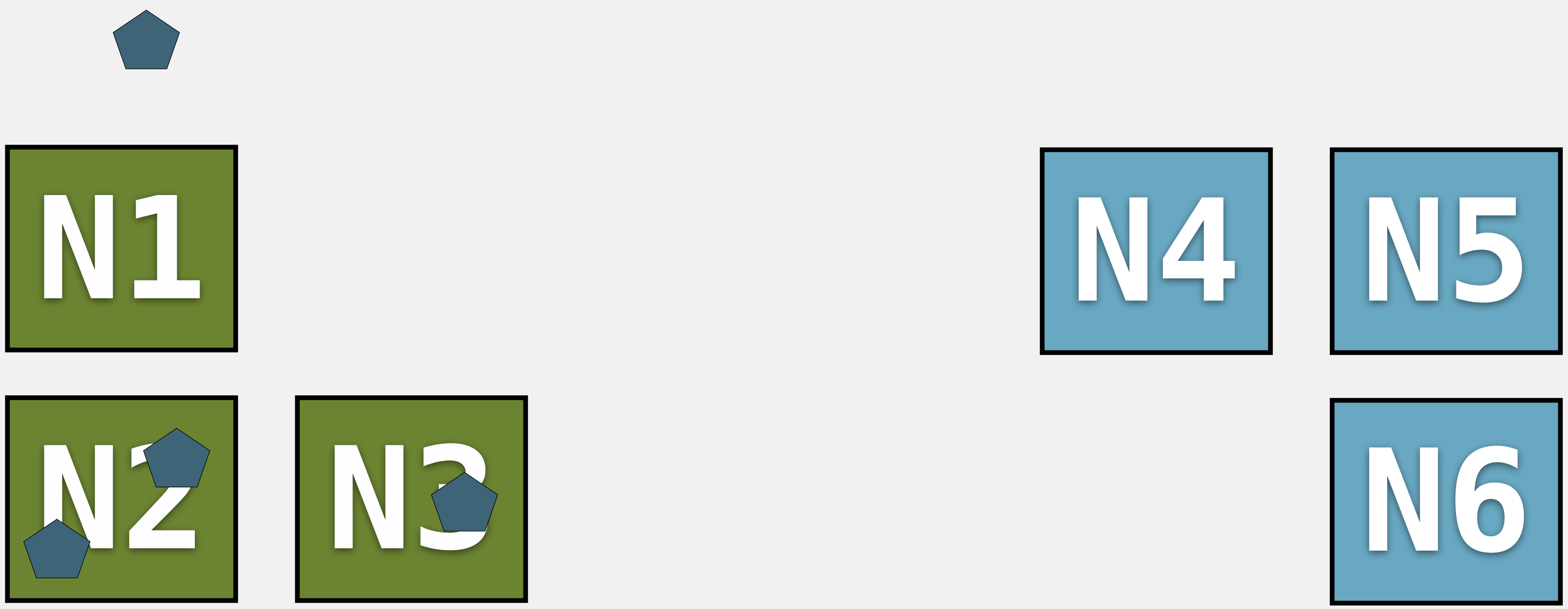


we must address
a scheduling problem

Continuous placement constraints with BtrPlace

Variables related to VM Management	
c^{host}	Current host of the VM (constant)
c^{men}, c^{cpu}	Current amount of memory and uCPU resources allocated to the VM (constant)
c^{ed}	Time the VM may leave its current host
d^{host}	Next host of the VM
d^{men}, d^{cpu}	Next amount of memory and uCPU resources to allocate to the VM
d^{st}	Time the VM arrives on its next host
Variables related to server management	
n^{q}	Next state of the server
Variables related to action management	
a^{st}, a^{ed}	Times an action starts and ends, respectively

from discrete to continuous among|simpleAmong

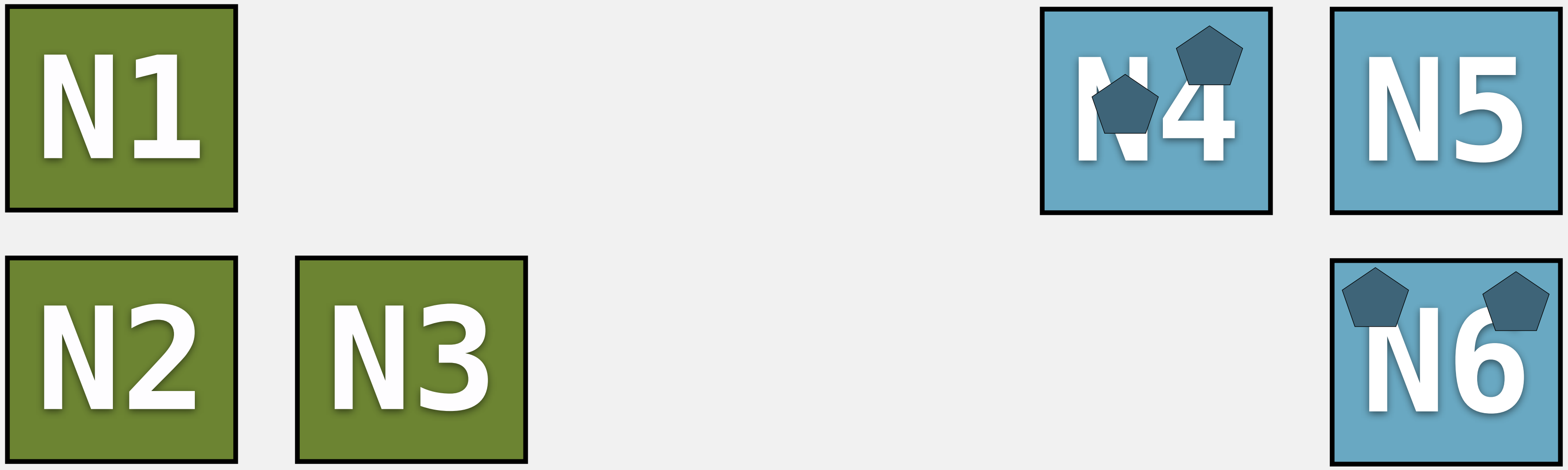


stay on a same partition by the end of the reconfiguration process

from discrete to continuous among|simpleAmong

stay on a same partition by the end of the reconfiguration process

from discrete to continuous
among|simpleAmong



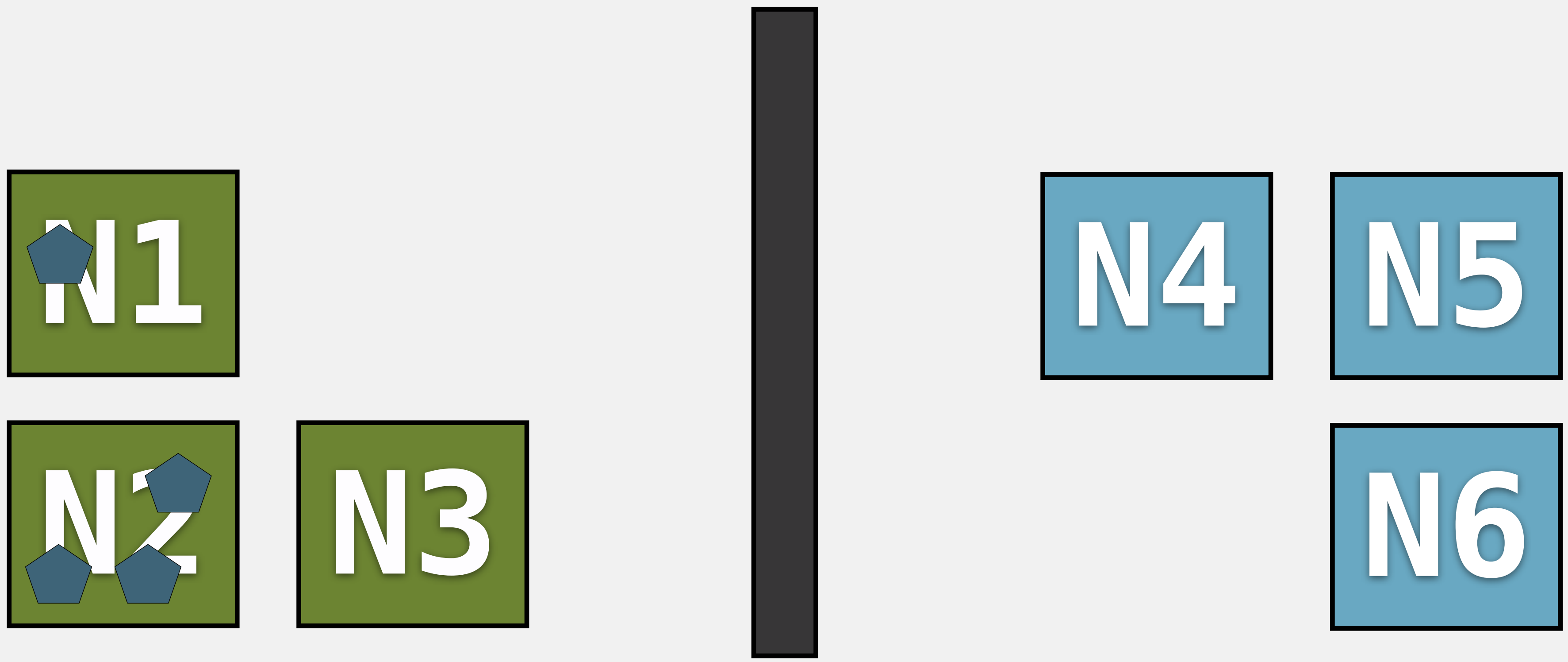
stay on a same partition by the end of the reconfiguration process

from discrete to continuous
among|simpleAmong

Disallow movements between partitions

- basic knowledge of a reconfiguration process
- still an assignment problem

from discrete to continuous
among|simpleAmong



Disallow movements between partitions

- basic knowledge of a reconfiguration process
- still an assignment problem

continuous spread

```
discrete spread(VM[1,2]) ::=
```

$$allDifferent(d_1^{host}, d_2^{host})$$

```
continuous spread(VM[1,2]) ::=
```

$$allDifferent(d_1^{host}, d_2^{host}) \wedge$$
$$d_1^{host} = c_2^{host} \implies a_1^{start} \geq a_2^{end} \wedge$$
$$d_2^{host} = c_1^{host} \implies a_2^{start} \geq a_1^{end}$$

Disallow temporary overlapping

- require to know this may happen
- scheduling 101

continuous maxOnline

```
discrete maxOnline(N[1..10], 7)::=
```

$$\sum_{i=1}^{10} n_i^q \leq 7$$

detailed knowledge of a reconfiguration process

```
continuous maxOnline(N[1..10], 7)::=
```

$$\forall i \in [1,10], \qquad n_i^{on} = \begin{cases} 0 & \text{if } n_i^q = 1 \\ a_i^{start} & \text{otherwise} \end{cases}$$

$$n_i^{off} = \begin{cases} max(T) & \text{if } n_i^q = 0 \\ a_i^{end} & \text{otherwise} \end{cases}$$

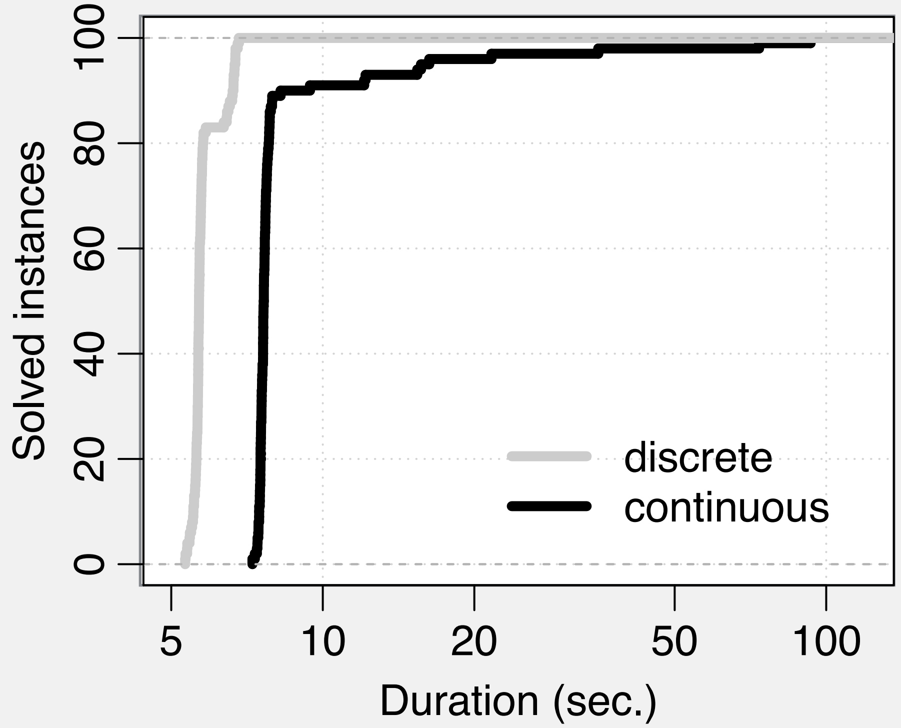
$$\forall t \in T, card(\{i | n_i^{on} \geq t \land n_i^{off}\}) \leq 7$$

scheduling 201

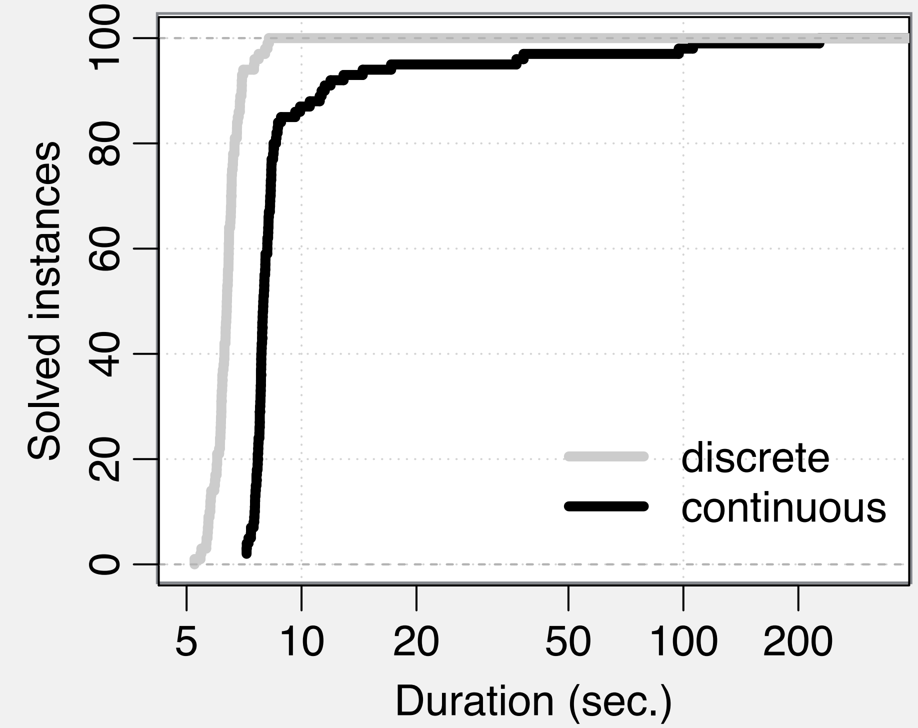
harder to imagine, model & implement

Performance overhead

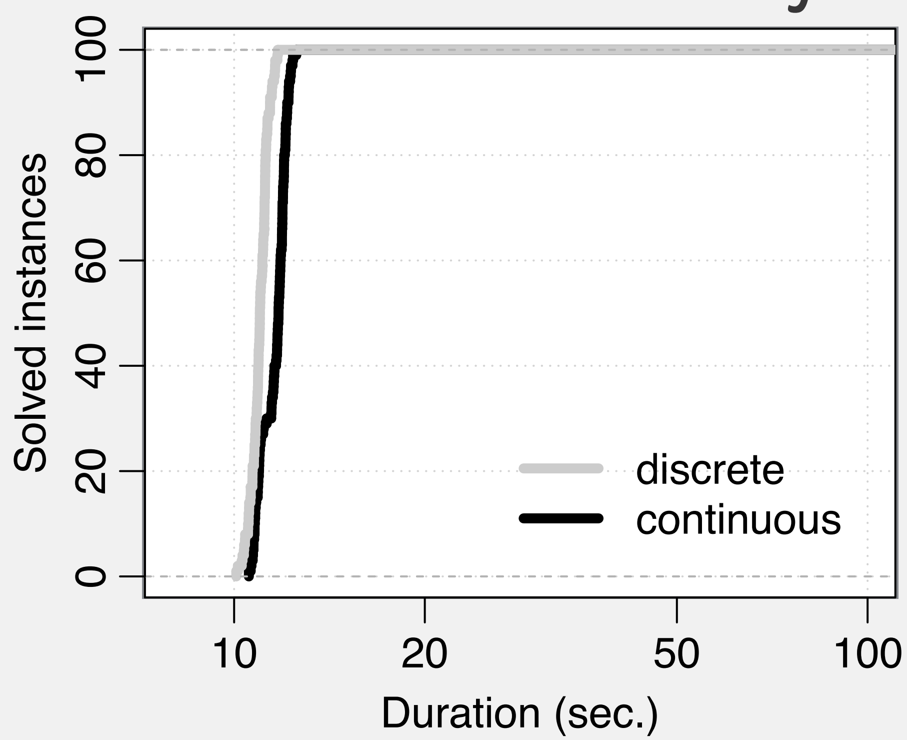
boot storm



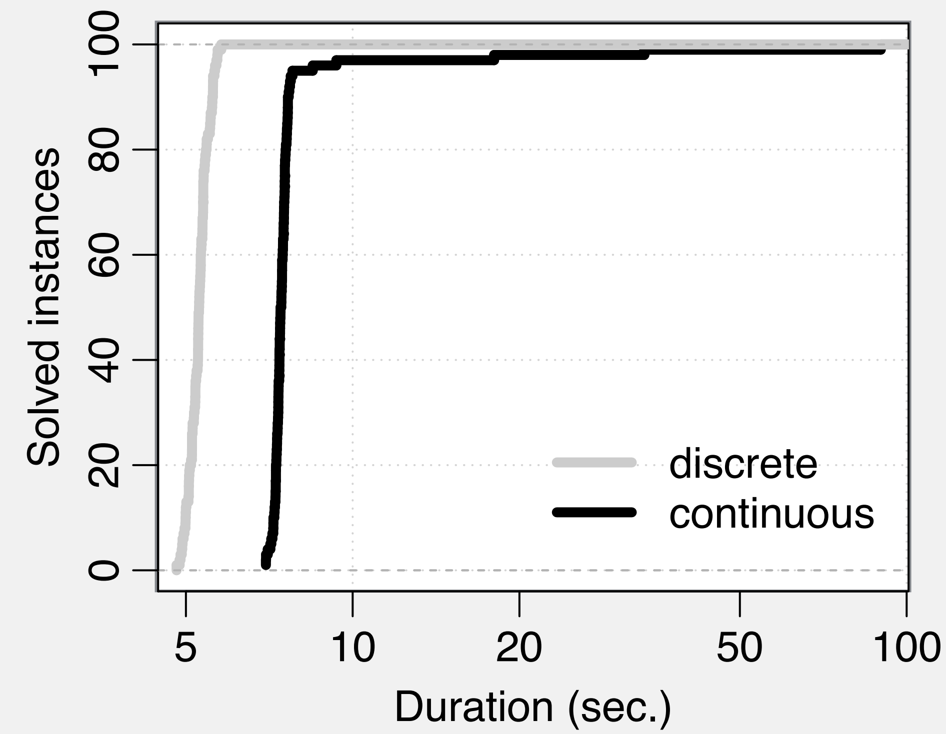
vertical elasticity



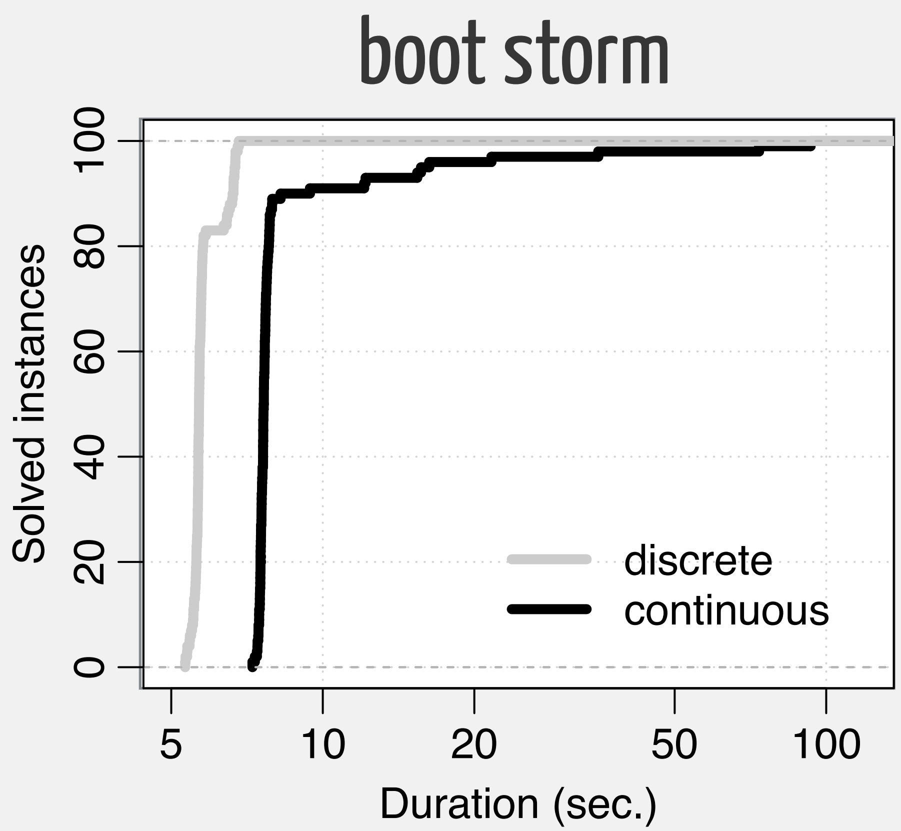
horitzontal elasticity

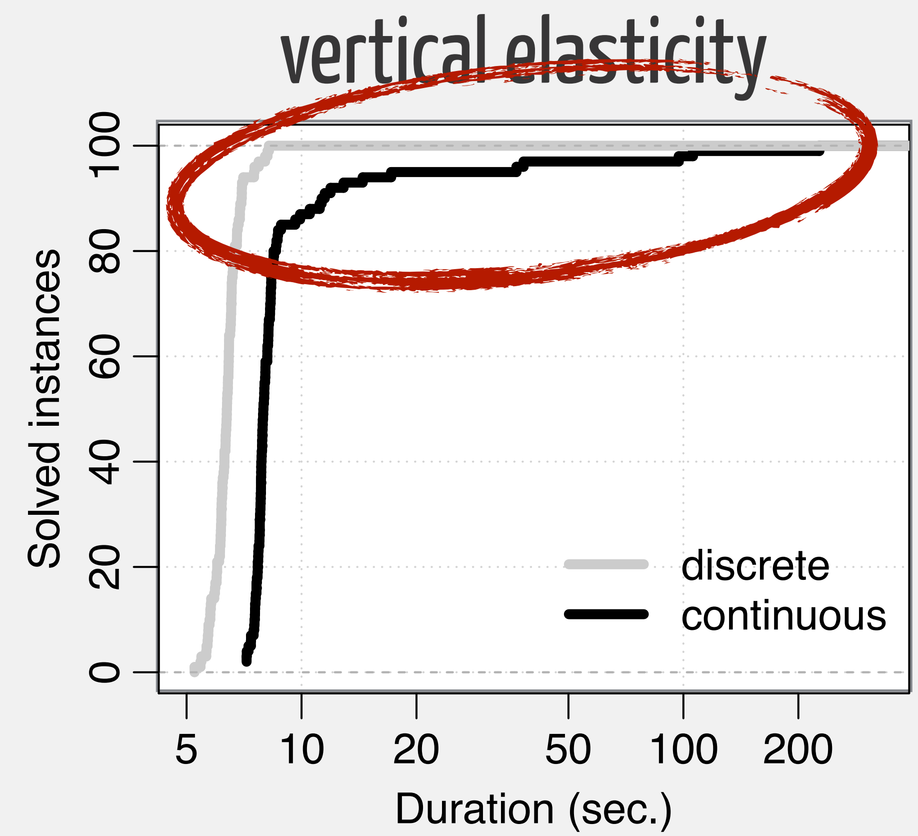


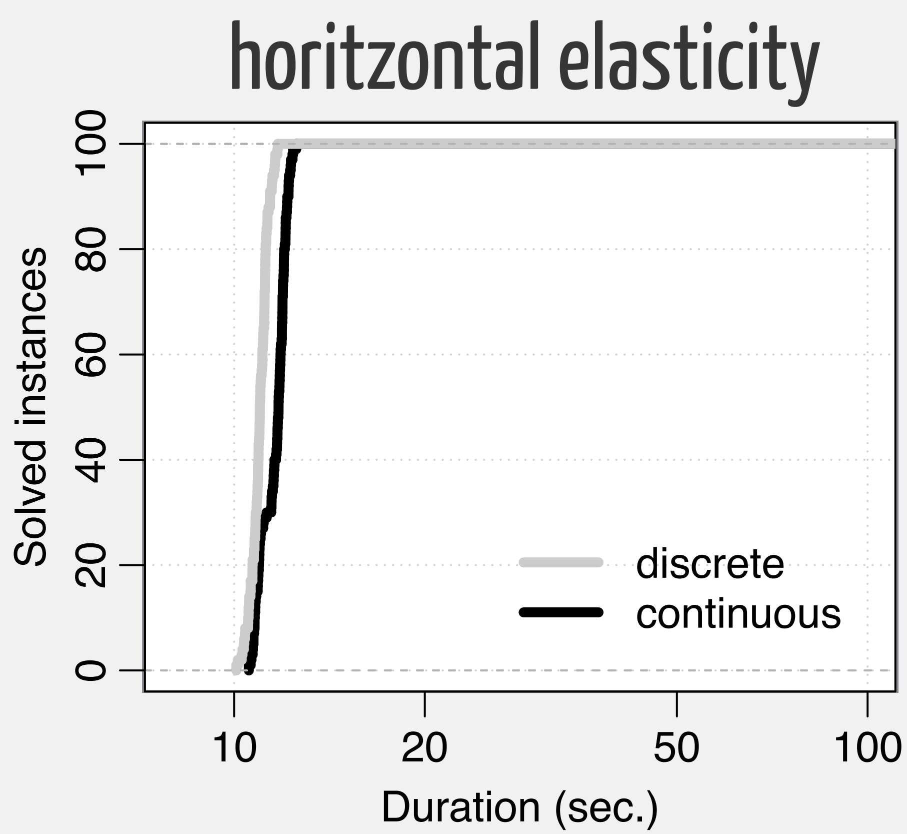
server failure

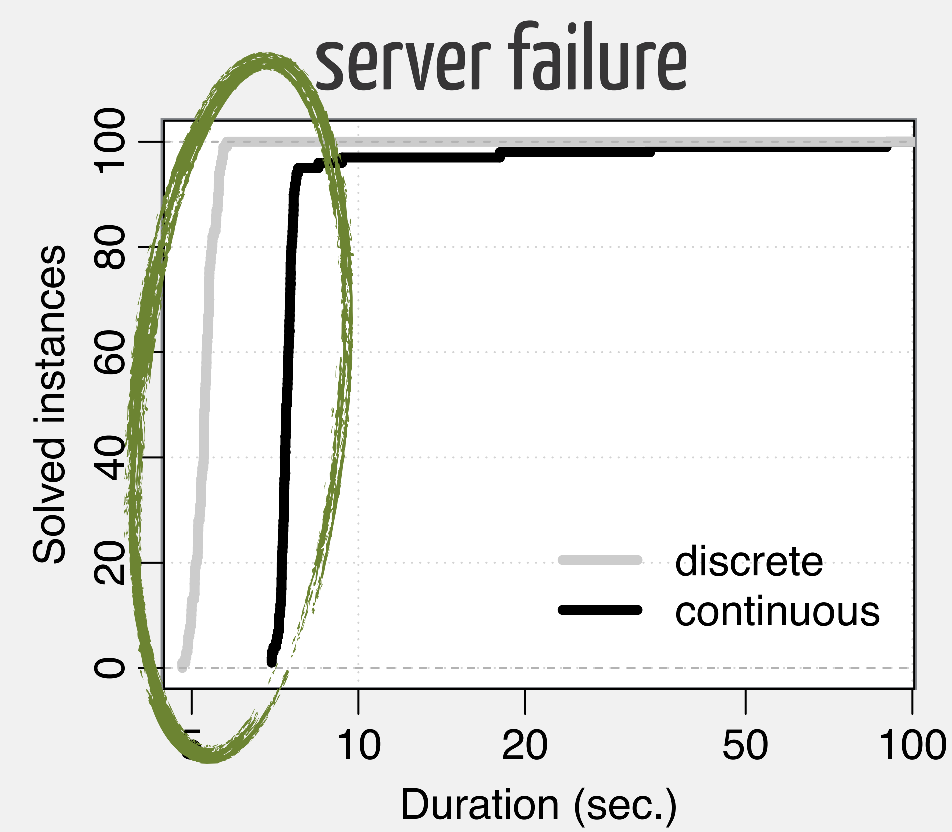


Performance overhead









Conclusions

- **discrete** restriction is not enough
- **continuous** restriction is a solution
- a **different view** on the problem
- **challenging**, but still possible to implement

Future Work

- a broader range of constraints and objectives
- reducing performance overhead
    - static analysis to detect un-necessary continuous constraints
    - controlled relaxation to handle hard situations

BtrPlace

http://btrp.inria.fr

open source, 20+ placement constraints,
demo, tutorials, everything for **reproducibility**